The Journal of Supercomputing, 17, 91–110, 2000

A Dynamic Diffusion Optimization Method for Irregular Finite Element Graph Partitioning

DON-LIN YANG dlyang@fcu.edu.tw
YEH-CHING CHUNG ychung@fcu.edu.tw
CHIH-CHANG CHEN cchen@fcu.edu.tw
CHING-JUNG LIAO cjliao@fcu.edu.tw

Department of Information Engineering, Feng Chia University, Taichung, Taiwan 407, ROC

Final version accepted March 16, 1999

Abstract. To efficiently execute a finite element application program on a distributed memory multicomputer, we need to distribute nodes of a finite element graph to processors of a distributed memory multicomputer as evenly as possible and minimize the communication cost of processors. This partitioning problem is known to be NP-complete. Therefore, many heuristics have been proposed to find satisfactory sub-optimal solutions. Based on these heuristics, many graph partitioners have been developed. Among them, Jostle, Metis, and Party are considered as the best graph partitioners available up-to-date. For these three graph partitioners, in order to minimize the total cut-edges, in general, they allow 3% to 5% load imbalance among processors. This is a tradeoff between the communication cost and the computation cost of the partitioning problem. In this paper, we propose an optimization method, the *dynamic diffusion method* (DDM), to balance the 3% to 5% load imbalance allowed by these three graph partitioners while minimizing the total cut-edges among partitioned modules. To evaluate the proposed method, we compare the performance of the dynamic diffusion method with the directed diffusion method and the multilevel diffusion method on an IBM SP2 parallel machine. Three 2D and two 3D irregular finite element graphs are used as test samples. For each test sample, 3% and 5% load imbalance situations are tested. From the experimental results, we have the following conclusions. (1) The dynamic diffusion method can improve the partition results of these three partitioners in terms of the total cut-edges and the execution time of a Laplace solver in most test cases while the directed diffusion method and the multilevel diffusion method may fail in many cases. (2) The optimization results of the dynamic diffusion method are better than those of the directed diffusion method and the multilevel diffusion method in terms of the total cut-edges and the execution time of a Laplace solver for most test cases. (3) The dynamic diffusion method can balance the load of processors for all test cases.

Keywords: distributed memory multicomputers, partition, mapping, load balancing, dynamic diffusion, irregular finite element graphs

1. Introduction

The finite element method is widely used for the structural modeling of physical systems. In the finite element model, an object can be viewed as a finite element graph, which is a connected and undirected graph that consists of a number of finite elements. Each finite element is composed of a number of nodes. The number of nodes of a finite element is determined by an application. In Figure 1, an example of a 21-node finite element model consisting of 24 finite elements is shown. Due to the

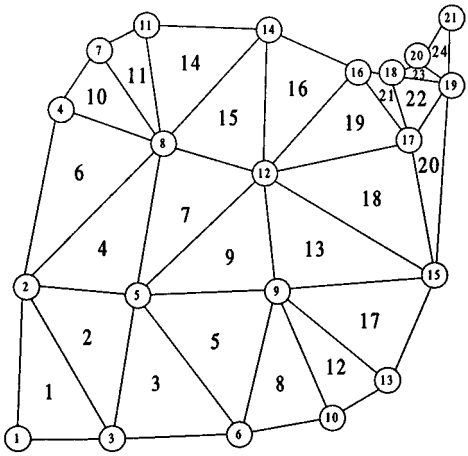


Figure 1. An example of a 21-node finite element graph with 24 finite elements (the circled and uncircled numbers denote the node numbers and finite element numbers, respectively).

properties of computation-intensiveness and computation-locality, it is very attractive to implement the finite element method on distributed memory multicomputers.

To efficiently execute a finite element application program on a distributed memory multicomputer, we need to map nodes of the corresponding finite element graph to processors of a distributed memory multicomputer such that each processor has the same amount of computational load and the communication among processors is minimized. Since this partitioning problem is known to be NP-complete [5], many heuristics have been proposed to find satisfactory sub-optimal solutions [1, 3, 4, 6, 8, 9, 12–15, 17–22]. Based on these heuristics, many graph partitioners have been developed [8, 12, 13, 17, 18, 21, 22]. Among them, Jostle [20], Metis [12], and Party [17] are considered as the best graph partitioners available up-to-date.

For these three graph partitioners, in order to minimize the total cut-edges, they allow 3% to 5% load imbalance among processors. This is a tradeoff between the communication cost and the computation cost in the partitioning problem. For some applications, the 3% to 5% load imbalance is acceptable. However, for most parallel finite element application programs, load balancing is a critical factor and can greatly affect the overall performance of application programs. In this paper, we propose an optimization method, the dynamic diffusion method (DDM), to efficiently balance the 3% to 5% load imbalance allowed by these three graph partitioners while minimizing the total cut-edges among partitioned modules. The dynamic diffusion method is designed to optimize the partitioning results of these three partitioners. Therefore, it is an offline (preprocessing) method. This method may not be suitable for adaptive mesh applications, that is, it is used as a dynamic load-balancing method. The reason is that it may fail to improve the overall performance of application programs when the load imbalance factor is greater than or equal to 10% among processors which usually occurs in adaptive mesh applications.

To evaluate the proposed method, we compare the performance of the dynamic diffusion method with the directed diffusion method and the multilevel diffusion

method on an IBM SP2 parallel machine. Three 2D and two 3D irregular finite element graphs are used as test samples. For each test sample, 3% and 5% load imbalance situations are tested. From the experimental results, we have the following conclusions. (1) The dynamic diffusion method can improve the partition results of these three partitioners in terms of the total cut-edges and the execution time of a Laplace solver in most test cases while the directed diffusion method and the multilevel diffusion method may fail in many cases. (2) The optimization results of the dynamic diffusion method are better than those of the directed diffusion method and the multilevel diffusion method in terms of the total cut-edges and the execution time of a Laplace solver for most test cases. (3) The dynamic diffusion method can balance the load of processors for all test cases.

The rest of the paper is organized as follows. Related work is presented in Section 2. In Section 3, the proposed dynamic diffusion method will be described in detail. In Section 4, the experimental results will be presented.

2. Related work

Many methods have been proposed in the literature to deal with the partitioning problems of irregular graphs onto distributed memory multicomputers. In general, they can be divided into five classes, the *orthogonal section* approach [19, 22], the *min-cut* approach [3, 4, 14], the *spectral* approach [1, 9, 19], the *multilevel* approach [1, 12, 13, 18, 22], and miscellaneous approaches [6, 15, 22]. These methods have been implemented in several graph partitioners, such as Chaco [8], DIME [22], Jostle [20], Metis [12], and Party [17], etc.

For the orthogonal section approach, an irregular graph is partitioned into modules and each module has the same amount of computational load. These modules are then assigned to processors. To partition an irregular graph, this approach recursively cut the graph into two subgraphs according to the coordinates of nodes on x-axis and y-axis in turn. Although this approach does not consider any connectivity information of the graph, it tries to group nodes that are close together in the graph to the same modules.

For the min-cut approach, the Kernighan-Lin heuristic [14] is the most frequently used method for local bisection. It uses a sequence of logical vertex pair exchange to determine the sets that have to be exchanged physically. Several heuristics are proposed to improve the performance of KL heuristic [4]. In [3], a recursive min-cut bipartitioning algorithm was proposed to map graphs onto hypercubes.

The spectral approach is based on algebraic graph theory. In this method, a matrix similar to the adjacency matrix of the graph is constructed and some specific eigenvectors of this matrix are determined. The determination of the eigenvectors is the major computational task of this method. According to the values of these eigenvectors, nodes of the graph are distributed to corresponding processors. Spectral methods are efficient in graph partitioning. However, the time and space required by the spectral methods to partition a graph are quite high.

The multilevel approach is based on a coarsening strategy that decreases the size of a graph in several levels using matching techniques. After the coarsening, it uses

the spectral method or a k-way partitioning method or other partitioning methods to partition the coarsening graph. After the partitioning, the partition of the coarse graph is extrapolated to the original one. Then the partitioned modules are assigned to processors.

Other methods such as the *index-based mapping* method [15], the *projection-based mapping* method [6], the *simulated annealing* (SA) method [22], etc., do not belong to the approaches described above.

If the partitioning method results in load imbalance, many load-balancing algorithms can be used to balance the load of each processor. The *dimension exchange method* (DEM) is applied to application programs without geometric structure [2]. Ou and Ranka [16] proposed a *linear programming-based* method to solve the incremental graph partition problem. This method is not guaranteed to find a solution since it has scope limitation for the transferred nodes that may result in no solutions sometimes.

Hu and Blake [11] proposed a *directed diffusion* method that computes the diffusion solution by using an unsteady heat conduction equation while optimally minimizing the Euclidean norm of the data movement. They proved that the diffusion solution can be found by solving the linear equation. The diffusion solution λ is a vector with n elements. For any two elements λ_i, λ_j in λ, if $\lambda_i - \lambda_j$ is positive, then partition i needs to send $\lambda_i - \lambda_j$ nodes to partition j. Otherwise, partition j needs to send $\lambda_j - \lambda_i$ nodes to partition i. Heirich and Taylor [7] proposed a direct diffusive load-balancing method for scalable multicomputers. They derived a reliable and scalable load balancing method based on properties of the parabolic heat equation $u_t - \alpha\nabla^2 u = 0$.

Horton [10] proposed a multilevel diffusion method by recursively bisecting a graph into two subgraphs and balancing the load of the two subgraphs. This method assumes that the graph can be recursively bisected into two connected graphs.

Schloegel *et al*. [18] also proposed a multilevel diffusion scheme to construct a new partition of the graph incrementally. It contains three phases, a coarsening phase, a multilevel diffusion phase, and a multilevel refinement phase. These algorithms perform diffusion in a multilevel framework and minimize data movement without comprising the edge-cut. Their methods also include parameterized heuristics to specifically optimize edge-cut, total data migration, and the maximum amount of data migrated in and out of any processor.

Walshaw *et al*. [21] implemented a parallel partitioner and a directed diffusion repartitioner in Jostle that is based on the diffusion solver proposed by Hu and Blake [11]. They also developed a multilevel diffusion repartitioner in Jostle.

3. The dynamic diffusion method

The dynamic diffusion method is a diffusion algorithm. The main difference between the dynamic diffusion method and the directed diffusion method [11] is the use of the diffusion process. In the directed diffusion method, the diffusion process is performed from over-loaded processors to under-loaded processors. How to select an over-loaded processor to start the diffusion process and how to diffuse

nodes from the over-loaded processors to its neighbor processors are two important issues in this method. This method does not guarantee the load-balancing property.

In the dynamic diffusion method, the diffusion process is guided by the load of processors and the connectivity of processors in a processor graph. When a finite element graph is partitioned into modules that are assigned to processors, we can get a *processor graph* from the partitioned modules (The processor graph will be described in Section 3.1). The connectivity of processors is the connections of processors in a processor graph. In each diffusion step, a processor that satisfies certain conditions will be selected as the processor to start the diffusion process no matter whether the processor is an over-loaded or an under-loaded processor. If the removal of the selected processor will not break the processor graph into two or more components, the selected processor will be removed from the processor graph once its load is balanced. Based on the new processor graph (after the removal of the selected processor), another diffusion step can be carried out. In at most $P \times (P+1)/2$ diffusion steps, the load of processors can be balanced, where P is the number of processors used by a partitioning method. Since the load of processors and the connectivity of processors are changed from a diffusion step to another, that is why the term *dynamic* comes. The dynamic diffusion method can be divided into the following four phases.

Phase 1: Construct a processor graph G from the partitioning result obtained from Jostle, Metis, or Party.
Phase 2: Calculate the under-load and over-load weights of processors in G.
Phase 3: Determine the load transfer sequence by using the dynamic diffusion heuristic.
Phase 4: Perform the load transfer and use the KL heuristic to reduce the cut-edges of boundary nodes.

In the following, we will describe the details of the algorithm. We assume that an N-node finite element graph and a P-processor distributed memory multicomputer are given.

3.1. The processor graph

When nodes of an irregular finite element graph are distributed to processors by Jostle, Metis, or Party, according to the communication property of the irregular finite element graph, we can get a processor graph from the partition. In a processor graph, nodes represent the processors and edges represent the communication needed among processors. The weights associated with nodes and edges denote the computation and the communication costs, respectively. We now give an example to explain it.

Example 1. Figure 2 shows an example of a processor graph. Figure 2(a) shows an initial partition of a 128-node irregular finite element graph on 10 processors by using the MLkP method (Metis). In Figure 2(a), the number of nodes assigned

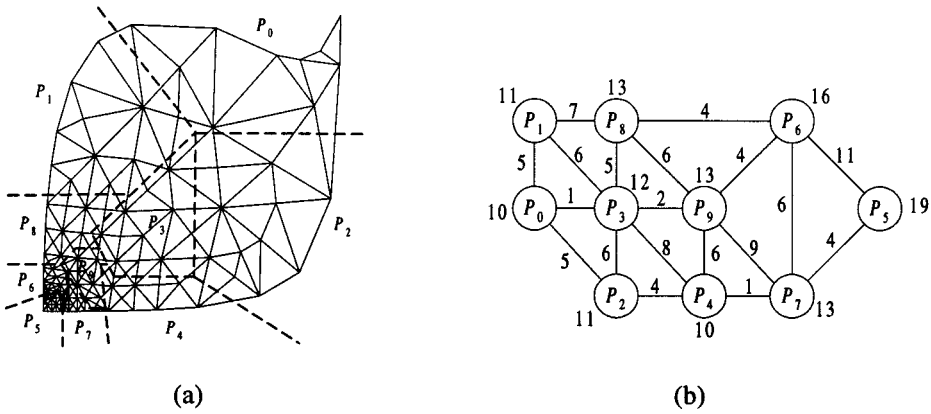


Figure 2. An example of a processor graph. (a) A partition of an irregular finite element graph. (b) The corresponding processor graph obtained from (a).

to processors $P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8$, and P_9 are 10, 11, 11, 12, 10, 19, 16, 13, 13, and 13, respectively. The corresponding processor graph of Figure 2(a) is shown in Figure 2(b).

3.2. *Calculate the under-load and over-load weights of processors*

In a processor graph, let $load(P_i)$ denote the number of nodes assigned to processor P_i, $quota(P_i) = N/P$ is the balanced load for processor P_i, and $weight(P_i) = load(P_i) - quota(P_i)$. If $load(P_i) > quota(P_i)$, processor P_i is called an *over-loaded* processor and the value of $weight(P_i)$ is called the *over-loaded weight* of processor P_i. If $load(P_i) < quota(P_i)$, processor P_i is called an *under-loaded* processor and the value of $weight(P_i)$ is called the *under-loaded weight* of processor P_i. Otherwise, P_i is called a *balanced* processor. Based on the processor graph obtained in phase 1 and above definitions, we can calculate the under-loaded and over-loaded weights of processors. Table 1 shows the under-loaded and over-loaded weights of the processors shown in Figure 2.

3.3. *Determine a load transfer sequence by using the dynamic diffusion heuristic*

A load transfer sequence for processors can be determined by using the dynamic diffusion heuristic. Assume that a processor graph $G = (V, E)$ is given. The dynamic diffusion heuristic is performed as follows:

Step 1: Sort processors in G according to their degrees in ascending order.
Step 2: Starting from the processor with the smallest degree, find processor P_v that satisfies the following conditions. If there are two or more candidate

Table 1. The under-loaded/over-loaded weights of processors shown in Figure 2

Processor	*load*	*quota*	*weight* (under-loaded/over-loaded)
0	10	13	−3
1	11	13	−2
2	11	13	−2
3	12	13	−1
4	10	13	−3
5	19	13	6
6	16	13	3
7	13	13	0
8	13	12	1
9	13	12	1

processors, select the processor that has the smallest under-loaded/over-loaded weight of candidate processors.

(a) The removal of P_v from G will not break G into two or more components.

(b) P_v is an over-loaded/balanced processor or if P_v is an under-loaded processor, the load of one of its neighboring processors is greater than its under-loaded weight.

Step 3: If no processor satisfies the conditions listed in Step 2, then select processor P_v that has the largest load in G. Mark processor P_v.

Step 4: If P_v is an under-loaded processor, then P_v needs to receive $|load(P_v) - quota(P_v)|$ nodes from its neighbor processor P_u, where P_u is one of the neighbor processors of P_v with the largest load. Record the send/receive relation and update the values of $load(P_v)$ and $load(P_u)$.

Step 5: If P_v is an over-loaded processor, then P_v needs to send $load(P_v) - quota(P_v)$ nodes to its neighbor processor P_w, where P_w is one of its neighbor processors with the smallest load and P_w has not been marked. Record the send/receive relation and update the values of $load(P_v)$ and $load(P_w)$.

Step 6: If the removal of processor P_v from G will not break G into two or more components, then remove P_v from G and clear all marks of processors.

Step 7: Perform Step 1 to Step 6 until the load of processors is balanced.

In the dynamic diffusion method, it tries to balance the load of one processor in each diffusion step. We have two possible cases. The first case is that we can find a processor P_v that satisfies the conditions listed in Step 2. In Step 2, if condition (a) is not satisfied, the removal of P_v from G will break G into two or more components. The consequence is that the dynamic diffusion method may not be able to balance the load of processors because processors can only transfer nodes to processors in the same component. Condition 2(b) ensures that a processor can receive the desired nodes from one of its neighbors if it is an under-loaded processor. If processor P_v satisfies the conditions listed in Step 2, we can balance the load of P_v

in current diffusion step. Once the load of P_v is balanced, it is deleted from the processor graph and will not be involved in the subsquent diffusion steps.

The second case is that we cannot find a processor that satisfies the conditions listed in Step 2. Therefore, the processor P_v selected in Step 3 is an over-loaded processor and the removal of P_v from G will break G into two or more componets. In this case, the load of P_v will be balanced in Step 5 and P_v will not be deleted from G in Step 6. Whenever a processor is marked, the load of the processor is balanced. Therefore, it is impossible to mark all processors in G; otherwise the load of processors will be balanced. It turns out that we can find a processor P_x that satisfies the conditions listed in Step 2 after several processors were marked. Then, the load of P_x can be balanced and P_x can be removed from G in the next diffusion step. For a given processor graph with P processors, at most $P-1$ processors can be marked. In the worst case, after $P \times (P+1)/2$ diffusion steps, the load of processors can be balanced. We now give an example to show the behavior of the dynamic diffusion heuristic.

Example 2. Figure 3 shows a step by step example of determining the load transfer sequence by using the dynamic diffusion heuristic for the processor graph shown in Figure 3(a). Initially, the degrees of processors P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, and P_{12} are 2, 5, 4, 3, 4, 2, 3, 5, 2, 2, 2, 4, and 2, respectively. The values of *load, quota,* and *weight* (under-loaded or over-loaded) of processors shown in Figure 3(a) are given in Table 2. When apply the dynamic diffusion heuristic to the processor graph shown in Figure 3(a), processors P_5, P_8, P_9, P_{10}, and P_{12} satisfy the conditions listed in Step 2. Since P_{10} has the smallest under-loaded/over-loaded weight among these five candidate processors, P_{10} is selected. Since $weight(P_{10}) = 0$, no send/receive relation is recorded in this diffusion step and P_{10} is removed from Figure 3(a). We obtain a new processor graph shown in Figure 3(b). Figure 3(c) to Figure 3(g) can be obtained in a similar manner as that of Figure 3(b).

When apply the dynamic diffusion heuristic to the processor graph shown in Figure 3(g), no processor satisfies the conditions listed in Step 2. Since P_6 has the largest load among processors, P_6 is selected as the candidate processor and is marked. P_6 has two neighbor processors, P_1 and P_{11}. Since P_1 has the smallest load in the neighbor processors of P_6 and P_1 has not been marked, a send/receive relation, $P_6 \xrightarrow{49} P_1$, is recorded in this diffusion step. Since P_6 does not satisfy the conditions listed in Step 2, P_6 is not removed from the processor graph shown in Figure 3(g). We obtain the processor graph shown in Figure 3(h). From Figure 3(h), we can see that if P_6 is removed from the processor graph, the load of processors cannot be balanced. Continuing to apply the dynamic diffusion heuristic to the processor graph shown in Figure 3(h), processor P_0 satisfies the conditions listed in Step 2. A send/receive relation, $P_1 \xrightarrow{8} P_0$, is recorded in this diffusion step, P_0 is removed from Figure 3(h), and all marks are cleared. We obtain the processor graph shown in Figure 3(i). Figure 3(j) to Figure 3(n) can be obtained in a similar manner as that of Figure 3(b). After the execution of the dynamic diffusion heuristic

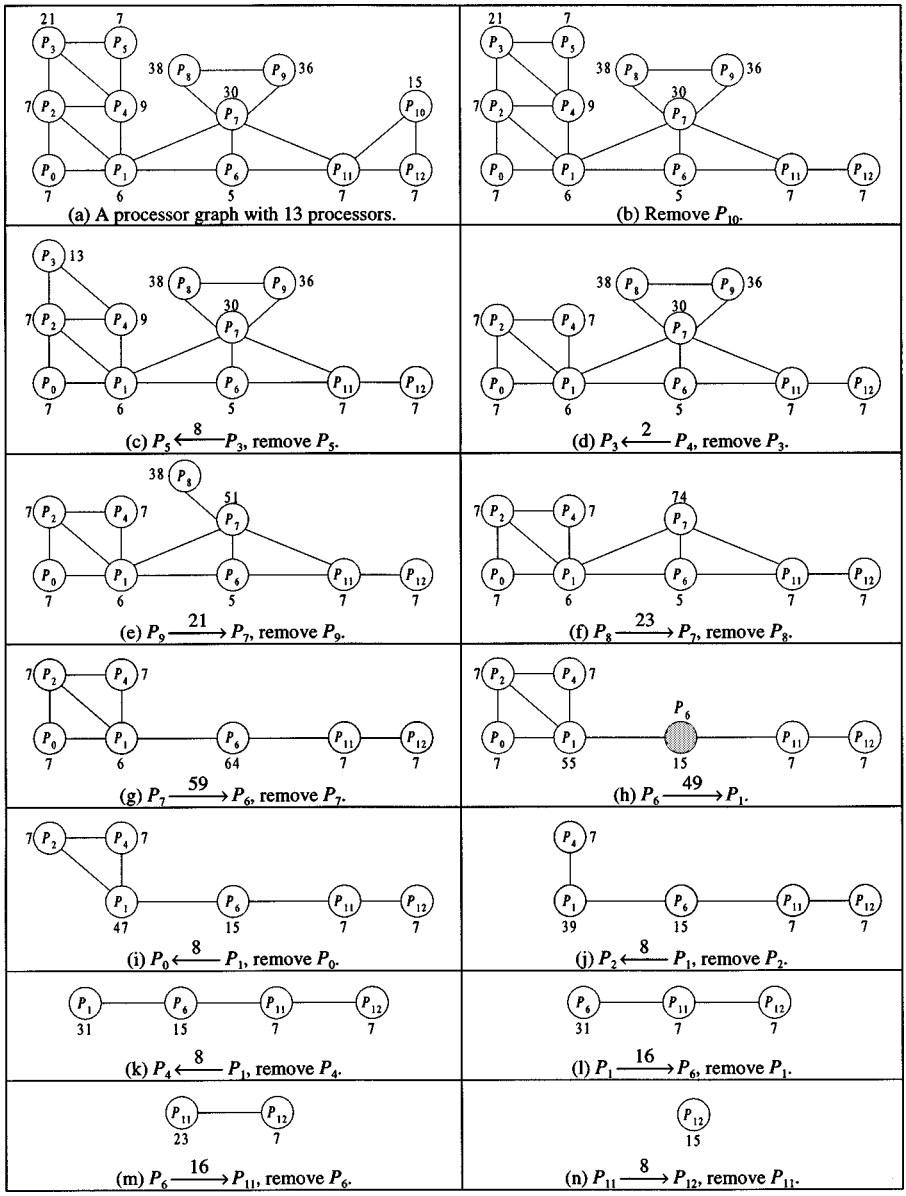


Figure 3. A step by step determination of the load transfer sequence by using the dynamic diffusion heuristic for the processor graph shown in (a).

is terminated, we obtain the following load transfer sequence,

$$P_5 \xleftarrow{8} P_3, P_3 \xleftarrow{2} P_4, P_9 \xrightarrow{21} P_7, P_8 \xrightarrow{23} P_7, P_7 \xrightarrow{59} P_6, P_6 \xrightarrow{49} P_1, P_0 \xleftarrow{8} P_1,$$
$$P_2 \xleftarrow{8} P_1, P_4 \xleftarrow{8} P_1, P_1 \xrightarrow{16} P_6, P_6 \xrightarrow{16} P_{11}, \text{ and } P_{11} \xrightarrow{8} P_{12}.$$

Table 2. The under-loaded/over-loaded weights of processors shown in Figure 3(a)

Processor	*load*	*quota*	*weight* (under-loaded/over-loaded)
0	7	15	−8
1	6	15	−9
2	7	15	−8
3	21	15	6
4	9	15	−6
5	7	15	−8
6	5	15	−10
7	30	15	15
8	38	15	23
9	31	15	16
10	15	15	0
11	7	15	−8
12	7	15	−8

Theorem 1. *Given a processor graph* $G = (V, E)$ *and the values of load and quota of processors, the dynamic diffusion heuristic can balance the load of processors.*

Proof. When determining load transfer sequence, in each diffusion step, processor P_v selected falls into one of the following cases.

Case 1: P_v satisfies the conditions listed in Step 2.
Case 2: P_v is an over-loaded processor and the removal of P_v from G will break G into two or more components.

If processor P_v selected in each diffusion step falls into case 1, the load of processors can be balanced after $P - 1$ diffusion steps. If processor P_v selected in some diffusion steps falls into case 2, the load of P_v will be balanced in Step 5 and P_v will not be deleted from G in Step 6. Since it is impossible to mark all processors in G (otherwise the load of processors is balanced), we can find a processor P_x that satisfies the conditions listed in Step 2 after several processors were marked (a sequence of diffusion steps). Then, the load of P_x can be balanced, P_x can be removed from G, and all marks will be reset in next diffusion step. For a given processor graph with P processors, at most $P - 1$ processors can be marked. In the worst case, after $P \times (P + 1)/2$ diffusion steps, the load of processors can be balanced. ∎

3.4. Perform the load transfer and minimize the cut-edges of boundary nodes

After the determination of the load transfer sequence, the physical load transfer can be carried out among the processors according to the load transfer sequence. The goals of the physical load transfer are to balance the load of processors and to minimize the communication cost among processors. Assume that processor P_i needs to send m nodes to processor P_j. To minimize the communication cost

between processors P_i and P_j, P_i sends nodes that are adjacent to those in P_j (we call these nodes as boundary nodes) to P_j. If the number of boundary nodes is greater than m, nodes with smaller degrees will be sent from P_i and P_j. If the number of boundary nodes is less than m, the boundary nodes and nodes that are adjacent to the boundary nodes will be sent from P_i to P_j. After performing the physical load transfer, the KL heuristic [14] is applied to minimize the cut-edges of boundary nodes.

The algorithm of the dynamic diffusion method is given as follows.

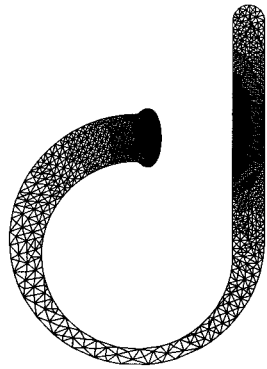
Algorithm Dynamic_Diffusion_Method(P) {
/* P is the number of processors */
/* Phase 1 */
1. Obtain a processor graph G from the partitioning result obtained from Jostle, Metis, or Party;
/* Phase 2 */
2. Calculate the under-load and over-load weights of processors in G;
/* Phase 3 */
3. Determine the load transfer sequence by using the dynamic diffusion heuristic;
/* Phase 4 */
4. **doparallel** {
5. Perform the load transfer according to the load transfer sequence;
6. Reduce the cut-edges of boundary nodes by using the KL heuristic; }

end_of_Dynamic_Diffusion_Method

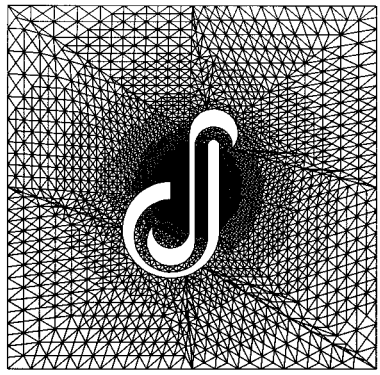
4. Performance evaluation and experimental results

To evaluate the performance of the dynamic diffusion method, three 2D and two 3D irregular finite element graphs are used as test samples. To obtain the three 2D irregular finite element graphs, we use the distributed irregular mesh environment (DIME) [23] to generate the initial irregular finite element graphs. We then use our mesh refinement tool to get the desired graphs. The two 3D finite element graphs, human femur and tibia, are produced by using our auto mesh generation program on source images obtained from CT (computer tomography). These five irregular finite element graphs are shown in Figure 4. The number of nodes, the number of elements, and the number of edges of these five irregular finite element graphs are given in Table 3. For presentation purposes, the number of nodes, the number of elements, and the number of edges of the irregular finite element graphs shown in Figure 4 are less than those shown in Table 3.

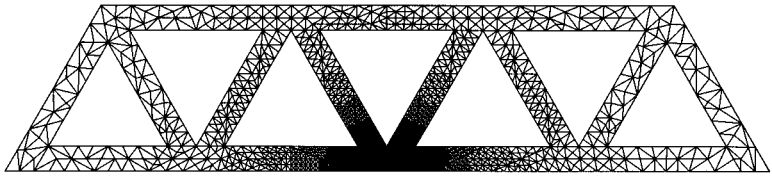
To get the experimental results, each test sample with 3% and 5% load imbalance cases on 10, 30, and 50 processors are tested. For each case, Jostle [20], Metis [12], and Party [17] are used to partition the test graph. Their partitioned results are then optimized by the dynamic diffusion method, the directed diffusion method (DD), and the multilevel diffusion method (MD). The programs of the directed diffusion method and the multilevel diffusion method used for the performance



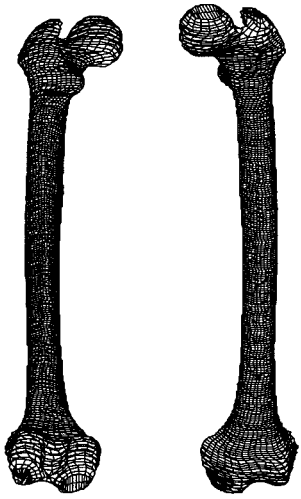
(a) *Hook* (1849 nodes, 3411 elements).



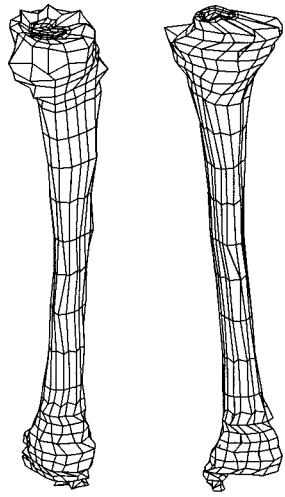
(b) *Letter* (6075 nodes, 11597 elements).



(c) *Truss* (7325 nodes, 14024 elements).



(d) *Femur* (6141 nodes, 7448 elements).



(e) *Tibia* (973 nodes, 1168 elements).

Figure 4. The test samples.

Table 3. The number of nodes, elements, and edges of the test samples

Samples	#nodes	#elements	#edges
Hook	80494	158979	239471
Letter	106215	126569	316221
Truss	57081	91968	169518
Femur	477378	953344	1430784
Tibia	557058	1114112	1671168

comparison were provided by Jostle [20]. By combining the partitioning methods and methods for optimization, there are twelve methods, denoted by $M_\phi = \{$Jostle, Metis, Party, Jostle/DD, Metis/DD, Party/DD, Jostle/MD, Metis/MD, Party/MD, Jostle/DDM, Metis/DDM, Party/DDM$\}$, used for the performance comparison. In M_ϕ, Jostle means that the partitioning method provided in Jostle is used to perform the partitioning without optimization. Metis/DDM means that the partitioning method provided in Metis is used to perform the partitioning and the dynamic diffusion method is used to optimize the partitioned result. In the following, we compare these methods in terms of the total cut-edges and the execution time of a Laplace solver of the partitioned/optimized results of test cases on an IBM SP2 parallel machine.

4.1. The total cut-edges

Table 4 to Table 8 show the total cut-edges and the maximum cut-edges of processors for test samples with 3% and 5% load imbalance on 10, 30, and 50 processors under different partitioning and/or optimization schemes. The total cut-edge is defined as the sum of the cut-edge of each processor. The maximum cut-edge is defined as the largest cut-edge of processors. Tables 4–6 show the total cut-edges and the maximum cut-edges of the three 2D test samples under different partitioning/optimization methods. In Tables 4–6, for Jostle, Metis, and Party, the dynamic

Table 4. The comparison of the total cut-edges and the maximum cut-edges for test sample *Hook*

	10 processors				30 processors				50 processors			
	3%		5%		3%		5%		3%		5%	
Method	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total
Jostle	233	1699	247	1732	162	4188	148	4255	142	6333	144	6214
Jostle/DD	233	1703	242	1748	162	4193	157	4215	141	6307	140	6189
Jostle/MD	229	1681	242	1745	160	4173	146	4157	142	6334	143	6125
Jostle/DDM	227	1644	230	1660	151	4085	148	4133	141	6020	144	6258
Metis	232	1716	210	1616	175	4381	175	4403	130	6296	118	6105
Metis/DD	229	1769	211	1687	175	4434	173	4439	128	6272	120	6193
Metis/MD	233	1760	217	1683	176	4391	173	4394	132	6190	118	6078
Metis/DDM	206	1610	206	1613	167	4362	150	4394	126	6367	117	6101
Party	236	1758	236	1789	176	4774	176	4745	157	6534	143	6480
Party/DD	227	1759	238	1797	176	4781	176	4778	157	6566	166	6392
Party/MD	228	1775	239	1791	181	4768	169	4739	151	6437	141	6533
Party/DDM	218	1743	221	1778	176	4800	167	4714	155	6501	143	6382

Table 5. The comparison of the total cut-edges and the maximum cut-edges for test sample *Letter*

	10 processors				30 processors				50 processors			
	3%		5%		3%		5%		3%		5%	
Method	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total
Jostle	270	2110	281	2005	187	5187	161	4966	159	7246	149	7114
Jostle/DD	270	2107	291	2062	187	5192	161	5077	159	7279	148	7154
Jostle/MD	266	2084	273	1995	187	5133	155	5054	153	7179	150	7132
Jostle/DDM	249	1919	267	1955	167	4934	152	4785	144	6949	147	6886
Metis	271	2020	303	2052	211	5408	247	5225	176	7311	147	7491
Metis/DD	275	2078	299	2121	210	5423	240	5330	175	7341	147	7551
Metis/MD	277	2052	295	2063	200	5296	182	5103	175	7252	140	7450
Metis/DDM	234	1878	248	1906	200	5145	184	5035	155	7025	144	7408
Party	325	2712	326	2711	250	5502	249	5490	164	7822	160	7753
Party/DD	325	2721	326	2718	250	5522	249	5517	171	7838	160	7785
Party/MD	322	2504	326	2697	245	5354	244	5483	162	7770	156	7724
Party/DDM	325	2634	326	2714	250	5466	244	5450	157	7750	153	7658

diffusion method and the multilevel diffusion method can improve their total cut-edges and maximum cut-edges in most test cases. The directed diffusion method fails to improve their total cut-edges and maximum cut-edges in many test cases. Moreover, the total cut-edges and the maximum cut-edges produced by the dynamic diffusion method are less than those of the directed diffusion method and the multilevel diffusion method in most test cases. For the three partitioning methods, Jostle produces the smallest total cut-edges and the smallest maximum cut-edges in most test cases.

Tables 7 and 8 show the total cut-edges and the maximum cut-edges of the two 3D test samples under different partitioning and/or optimization schemes. In Tables 7 and 8, for Jostle and Metis, the dynamic diffusion method can improve their total cut-edges and maximum cut-edges for all test cases. However, the directed diffusion method and the multilevel diffusion method fail to improve their total cut-edges and maximum cut-edges in some test cases. The total cut-edges and the maximum cut-

Table 6. The comparison of the total cut-edges and the maximum cut-edges for test sample *Truss*

	10 processors				30 processors				50 processors			
	3%		5%		3%		5%		3%		5%	
Method	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total
Jostle	218	2073	184	2053	121	4279	125	4387	107	5955	99	5753
Jostle/DD	218	2073	184	2050	122	4314	123	4395	107	5953	96	5792
Jostle/MD	223	2111	180	2049	121	4255	115	4327	106	5875	96	5732
Jostle/DDM	202	1960	177	1924	109	4031	116	4214	102	5787	97	5614
Metis	209	2267	212	2146	143	4520	153	4405	105	5895	114	5930
Metis/DD	209	2265	213	2171	143	4533	154	4440	105	5896	116	5956
Metis/MD	195	2238	191	2111	140	4510	154	4399	101	5839	114	5931
Metis/DDM	191	2158	184	2029	129	4276	129	4193	101	5676	109	5667
Party	265	2435	258	2404	135	4686	140	4687	148	5916	148	5902
Party/DD	265	2438	258	2412	135	4710	140	4722	147	5948	147	5947
Party/MD	274	2427	258	2404	156	4706	135	4670	97	5874	100	5959
Party/DDM	267	2332	252	2343	133	4650	132	4634	94	5658	96	5638

Table 7. The comparison of the total cut-edges and the maximum cut-edges for test sample *Femur*

	10 processors				30 processors				50 processors			
	3%		*5%*		*3%*		*5%*		*3%*		*5%*	
Method	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total
Jostle	634	5480	628	5286	440	15146	392	14907	310	20067	285	19353
Jostle/DD	634	5478	628	5284	440	15123	392	14927	310	19897	286	19405
Jostle/MD	632	5462	628	5280	440	15062	386	14879	310	20075	285	19384
Jostle/DDM	624	5148	618	5122	398	14304	371	14330	308	18826	276	18798
Metis	872	5916	786	6012	430	14971	433	14825	324	19850	308	19688
Metis/DD	880	5976	762	5870	430	14967	435	14916	324	19848	308	19682
Metis/MD	838	6074	766	6100	429	14917	408	15056	318	19817	308	19662
Metis/DDM	736	5184	734	5710	415	14462	388	13976	314	19548	290	18878
Party	514	4626	514	4624	372	13973	372	14169	267	18856	267	18891
Party/DD	516	4640	516	4638	372	13989	372	14198	267	18913	267	18964
Party/MD	516	4642	518	4644	372	13973	372	14176	271	18914	270	18967
Party/DDM	516	4638	516	4634	349	13798	366	14048	280	19173	262	18626

edges produced by the dynamic diffusion method are less than those of the directed diffusion method and multilevel diffusion method for all test cases. For Party, the dynamic diffusion method can improve its total cut-edges and maximum cut-edges in most test cases. However, the directed diffusion method and the multilevel diffusion method fail to improve its total cut-edges and maximum cut-edges in most test cases. The total cut-edges and the maximum cut-edges produced by the dynamic diffusion method are less than those of the directed diffusion method and multilevel diffusion method in most test cases. For the three partitioning methods, Party produces the smallest total cut-edges and the smallest maximum cut-edges for all test cases.

4.2. The execution time of a Laplace solver

To map an N-node finite element graph onto a P-processor distributed memory multicomputer, we need to assign nodes of the graph to processors of the multi-

Table 8. The comparison of the total cut-edges and the maximum cut-edges for test sample *Tibia*

	10 processors				30 processors				50 processors			
	3%		*5%*		*3%*		*5%*		*3%*		*5%*	
Method	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total	Max	Total
Jostle	743	9257	698	9057	528	17553	424	16932	411	22620	338	21922
Jostle/DD	743	9257	698	9057	528	17542	424	16930	411	22614	338	21887
Jostle/MD	741	9195	690	9032	524	17364	443	16790	397	22404	326	21738
Jostle/DDM	690	8337	675	8655	420	15403	377	15651	353	20495	299	19719
Metis	805	9559	818	9372	519	17749	530	17091	386	22555	385	22391
Metis/DD	805	9559	818	9372	519	17745	530	17091	384	22530	383	22388
Metis/MD	801	9549	818	9362	514	17532	507	17069	370	22395	383	22320
Metis/DDM	695	8353	741	8346	499	15682	399	15423	273	19945	339	20325
Party	696	8639	682	8664	414	16064	444	16069	339	21403	331	21222
Party/DD	697	8644	684	8677	414	16085	445	16116	341	21458	330	21302
Party/MD	726	8902	715	8781	412	16068	444	16070	338	21559	331	21308
Party/DDM	686	8393	670	8279	397	15445	439	15469	342	20601	328	20868

computer. There are P^N mappings. The execution time of a finite element graph on a distributed memory multicomputer under a particular partitioning/optimization method L_i can be defined as follows:

$$T_{par}(L_i) = \max\{T_{comp}(L_i, P_j) + T_{comm}(L_i, P_j)\}, \tag{1}$$

where $T_{par}(L_i)$ is the execution time of a finite element application program on a distributed memory multicomputer under L_i, $T_{comp}(L_i, P_j)$ is the computation cost of processor P_j under L_i, and $T_{comm}(L_i, P_j)$ is the communication cost of processor P_j under L_i, where $i = 1, ..., P^N$ and $j = 0, ..., P - 1$.

The cost model used in Equation 1 is assuming a synchronous communication mode in which each processor goes through a computation phase followed by a communication phase. Therefore, the computation cost of processor P_j under a partitioning/optimization method L_i can be defined as follows:

$$T_{comp}(L_i, P_j) = S \times load_i(P_j) \times T_{task}, \tag{2}$$

where S is the number of iterations performed by a finite element method, $load_i(P_j)$ is the number of nodes of a finite element graph assigned to processor P_j, and T_{task} is the time for a processor to execute a task.

In general, it is possible to overlap the communication with the computation. In this case, $T_{comm}(L_i, P_j)$ may not always reflect the true communication cost since it could be partially overlapped with that of the computation. However, $T_{comm}(L_i, P_j)$ can provide a good estimate for the communication cost. Since we use a synchronous communication mode, $T_{comm}(L_i, P_j)$ can be defined as follows:

$$T_{comm}(L_i, P_j) = S \times (\delta \times T_{setup} + \phi \times T_c), \tag{3}$$

where S is the number of iterations performed by a finite element method, δ is the number of processors that processor P_j has to send data to in each iteration, T_{setup} is the setup time of the I/O channel, ϕ is the total number of bytes that processor P_j has to send out in each iteration, and T_c is the data transmission time of the I/O channel per byte.

In our experimental test, a finite element application program is a Laplace solver. Tables 9–13 show the time of a Laplace solver to execute one iteration (computation + communication) for test cases under different partitioning/optimization methods. Comparing Tables 4–8 and Tables 9–13, for the same test case, if the total cut-edge produced by a partitioning/optimization method L_i is less than that of another L_j, in general, the execution time of a Laplace solver for the test case under L_i is less than that of L_j. However, there are some exceptions. For example, for the test sample *Femur*, the total cut-edges produced by Party and Party/DDM are 4626 and 4638, respectively, for the case where 3% load imbalance and 10 processors are tested. The execution time of a Laplace solver for the test case under Party and Party/DDM is 737.694 and 737.302 milliseconds. The reason is that Party allows 3% load imbalance while Party/DDM fully balances the load of processors. The performance gain of the total cut-edge (the communication) of

Table 9. The time comparison of a Laplace solver to execute one iteration (computation+ communication) for test sample *Hook* (time unit: 1×10^{-3} second)

	10 processors		30 processors		50 processors	
Method	3%	5%	3%	5%	3%	5%
Jostle	143.967	149.744	64.677	65.032	56.364	47.800
Jostle/DD	144.687	147.205	66.126	65.461	55.528	46.512
Jostle/MD	144.030	145.596	64.277	62.473	56.259	47.214
Jostle/DDM	142.754	142.665	60.383	61.097	46.169	46.017
Metis	149.494	143.919	64.659	64.918	47.011	43.385
Metis/DD	147.774	142.346	65.330	63.585	48.251	48.041
Metis/MD	148.726	142.589	65.474	63.071	44.628	42.241
Metis/DDM	140.089	138.901	62.878	60.317	50.720	43.595
Party	143.779	144.514	71.496	70.830	61.555	57.736
Party/DD	144.555	145.105	69.985	73.479	60.583	57.582
Party/MD	144.253	144.620	70.201	70.625	54.717	55.676
Party/DDM	142.621	142.138	71.039	69.016	57.068	52.695

Party is offset by the 3% load imbalance (the computation). From this example, we can see that the load balancing is an important factor for the overall performance of a finite element application program on a distributed memory multicomputer.

5. Conclusions

In this paper, we have proposed a dynamic diffusion method to optimize the partitioning results of three partitioners, Jostle, Metis, and Party. For these three graph partitioners, in order to minimize the total cut-edges, in general, they allow 3% to 5% load imbalance among processors. This is a tradeoff between the communication cost and the computation cost of the partitioning problem. To evaluate the dynamic diffusion method, three 2D and two 3D irregular finite element graphs are

Table 10. The time comparison of a Laplace solver to execute one iteration (computation+ communication) for test sample *Letter* (time unit: 1×10^{-3} second)

	10 processors		30 processors		50 processors	
Method	3%	5%	3%	5%	3%	5%
Jostle	189.600	189.548	83.602	81.175	63.059	60.049
Jostle/DD	189.449	191.055	82.721	84.848	63.863	61.663
Jostle/MD	188.318	185.996	82.418	82.150	60.305	60.916
Jostle/DDM	183.259	185.266	73.162	73.925	52.717	53.912
Metis	190.485	194.287	87.856	91.855	64.274	57.089
Metis/DD	192.009	193.601	87.116	92.541	65.873	58.323
Metis/MD	189.752	191.690	81.950	80.108	62.324	55.134
Metis/DDM	180.614	181.738	77.699	76.551	55.400	52.685
Party	195.709	190.673	89.790	85.635	66.727	61.881
Party/DD	197.177	189.761	89.371	86.569	66.438	64.311
Party/MD	191.658	190.040	83.799	85.730	58.488	61.306
Party/DDM	192.993	190.679	86.401	83.221	59.259	57.816

Table 11. The time comparison of a Laplace solver to execute one iteration (computation+communication) for test sample *Truss* (time unit: 1×10^{-3} second)

	10 processors		30 processors		50 processors	
Method	3%	5%	3%	5%	3%	5%
Jostle	112.879	110.270	55.962	53.933	40.941	39.785
Jostle/DD	112.438	108.844	56.793	53.415	42.140	40.364
Jostle/MD	114.403	109.373	55.791	51.265	40.482	38.655
Jostle/DDM	107.637	104.634	48.332	47.353	36.731	35.837
Metis	111.879	113.619	58.341	57.525	43.324	45.798
Metis/DD	112.949	114.057	57.373	59.411	43.659	46.678
Metis/MD	110.538	109.853	56.584	56.768	40.190	46.874
Metis/DDM	106.363	107.066	49.365	49.432	35.269	38.065
Party	119.516	116.319	56.287	57.536	52.230	51.821
Party/DD	118.104	117.119	57.050	57.759	51.794	53.552
Party/MD	118.465	117.008	58.530	55.573	42.807	44.907
Party/DDM	115.165	115.499	54.296	53.317	35.580	35.233

used as test samples. For each test sample, 3% and 5% load imbalance situations are tested. We compare the performance of the dynamic diffusion method with the directed diffusion method and the multilevel diffusion method in terms of the total cut-edges and the execution time of a Laplace solver for the test samples on an IBM SP2 parallel machine. From the experimental results, we have the following conclusions.

1. The dynamic diffusion method can improve the partitioning results of these three partitioners in terms of the total cut-edges and the execution time of a Laplace solver in most test cases while the directed diffusion method and the multilevel diffusion method may fail in many cases.
2. The optimization results of the dynamic diffusion method are better than those of the directed diffusion method and the multilevel diffusion method in terms

Table 12. The time comparison of a Laplace solver to execute one iteration (computation+communication) for test sample *Femur* (time unit: 1×10^{-3} second)

	10 processors		30 processors		50 processors	
Method	3%	5%	3%	5%	3%	5%
Jostle	759.047	754.324	304.355	290.988	214.713	191.519
Jostle/DD	759.062	753.533	303.829	292.203	208.482	193.983
Jostle/MD	758.358	753.265	300.654	290.863	214.428	193.731
Jostle/DDM	749.339	747.962	274.690	272.789	175.536	173.163
Metis	803.828	803.167	292.515	303.963	187.649	201.726
Metis/DD	798.190	767.846	293.150	305.408	188.231	201.162
Metis/MD	796.600	773.730	292.584	307.310	185.608	199.893
Metis/DDM	760.287	759.359	277.461	273.821	177.603	174.247
Party	737.694	736.367	278.361	276.135	173.300	181.736
Party/DD	737.739	738.451	278.391	276.271	174.858	183.251
Party/MD	739.333	739.401	276.395	275.865	175.594	183.024
Party/DDM	737.302	737.412	270.134	270.709	178.754	173.322

Table 13. The time comparison of a Laplace solver to execute one iteration (computation+ communication) for test sample *Tibia* (time unit: 1×10^{-3} second)

	10 processors		30 processors		50 processors	
Method	3%	5%	3%	5%	3%	5%
Jostle	904.997	885.558	391.656	355.122	272.735	268.391
Jostle/DD	904.193	884.955	391.139	355.921	274.300	267.793
Jostle/MD	902.412	883.184	385.988	351.614	265.971	262.086
Jostle/DDM	871.809	870.054	316.324	311.341	203.858	198.662
Metis	920.364	915.677	389.514	379.094	285.797	268.843
Metis/DD	921.352	917.348	388.046	378.505	284.380	269.445
Metis/MD	919.571	915.816	381.623	376.092	278.865	268.104
Metis/DDM	874.137	879.595	324.007	315.079	195.532	201.867
Party	881.222	886.099	334.580	336.531	230.737	216.396
Party/DD	883.332	886.816	334.693	340.031	232.150	216.945
Party/MD	891.727	890.122	333.267	337.882	233.089	217.967
Party/DDM	871.196	869.823	313.723	318.067	205.151	205.033

of the total cut-edges and the execution time of a Laplace solver for most test cases.

3. The dynamic diffusion method can balance the load of processors for all test cases.

Acknowledgments

The work of this paper was partially supported by NSC of R.O.C. under contract NSC-87-2213-E-035-010. The authors would like to thank Dr. Robert Preis, Professor Karypis, and Professor Chris Walshaw for providing the Party, the Metis, and Jostle software packages.

References

1. S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, April 1994.
2. G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2):279–301, October 1989.
3. F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. *Journal of Parallel and Distributed Computing*, 10:35–44, 1990.
4. C. M. Fiduccia and R. M. Mattheyes. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pp. 175–181, 1982.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to Theory of NP-Completeness*. Freeman, San Francisco, Calif., 1979.
6. J. R. Gilbert, G. L. Miller, and S. H. Teng. Geometric mesh partitioning: implementation and experiments. In *Proceedings of the 9th International Parallel Processing Symposium*, Santa Barbara, Calif., pp. 418–427, April 1995.
7. A. Heirich and S. Taylor. A parabolic load balancing method. In *Proceedings of ICPP '95*, pp. 192–202, 1995.
8. B. Hendrickson and R. Leland. The Chaco user's guide: version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, October 1994.

9. B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.
10. G. Horton. A multi-level diffusion method for dynamic load balancing. *Parallel Computing*, 19:209–218, 1993.
11. Y. F. Hu and R. J. Blake. An optimal dynamic load balancing algorithm. Technical Report DL-P-95-011, Daresbury Laboratory, Warrington, UK, 1995.
12. G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, January 1998.
13. G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, January 1998.
14. B. W. Kernigham and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(2):292–370, February 1970.
15. C. W. Ou, S. Ranka, and G. Fox. Fast and parallel mapping algorithms for irregular problems. *The Journal of Supercomputing*, 10(2):119–140, December 1996.
16. C. W. Ou and S. Ranka. Parallel incremental graph partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):884–896, August 1997.
17. R. Preis and R. Diekmann. The PARTY partitioning—library user guide—version 1.1. Heniz Nixdorf Institute Universität Paderborn, Germany, September 1996.
18. K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing*, 47(2):109–124, December 1997.
19. H. D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2/3):135–148, 1991.
20. C. Walshaw. The Jostle user manual: version 2.0. University of Greenwich, London, UK, July 1997.
21. C. H. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, December 1997.
22. R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3(5):457–481, October 1991.
23. R. D. Williams. *DIME: Distributed Irregular Mesh Environment*. California Institute of Technology, 1990.